TEXT CLASSIFICATION USING PIPELINING
EEE F266

BY

AKASH CHINTHA 2014A3PS241P

INSTRUCTOR

ABHIJIT R ASATI

ELECTRICAL AND ELECTRONICS ENGINEERING

BITS-PILANI

TABLE OF CONTENTS

I. ABSTRACT

NLTK, the Natural Language Toolkit, is a suite of open source program modules providing ready_to_use computational linguistics courseware. Scikit_learn, a Machine Learning toolkit offers an extensive library of Machine Learning algorithms and techniques that simplify creation of working models and applications. Using the pipelining feature of scikit_learn and methods of NLTK, parsing, tokenizing, and vectorization of text is used to train a classifier which is then used to make predictions on a dataset; thus, making a useful application.

Keywords: NLTK, Scikit_learn, Pipelining, Classifier, Vectorization, Lemmatization.

II. FRAMEWORKS AND TOOLS

A) Scikit

Scikitlearn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k_means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

B) NLTK

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

III. SETTING UP THE TOOLS

Before settingup the tools required, we need to install Python 3 in the working OS. We need to make sure to include pip package manager while installing Python. Python 3 is recommended compared to Python 2.

A) Installing NLTK

By running the command $pip\ install - U\ nltk$, python installs NLTK library.

To import working data/models from NLTK, we need to run $nltk.download()$ command in python that prompts a downloader where we can select the models that are required to be downloaded.

Following are the imports needed:

From nltk.corpus import stopwords as sw

From nltk.corpus import wordnet as wn

From nltk import wordpunct_tokenize

From nltk import WordNetLemmatizer

From nltk import sent_tokenize

From nltk import pos_tag

B) Installing Scikit-learn

By running the command $pip\ install - U\ scikit - learn$, python installs scikit-learn library.

For scikitlearn to be fully functional, we also need to use ‘pip’ to install NumPy and SciPy too.

Following are the inputs needed:

From sklearn.pipeline import Pipeline

From sklearn.preprocessing import LabelEncoder

From sklearn.linear_model import SGDClassifier

From sklearn.base import BaseEstimator, TransformerMixin

From sklearn.metrics import classification_report as clsr

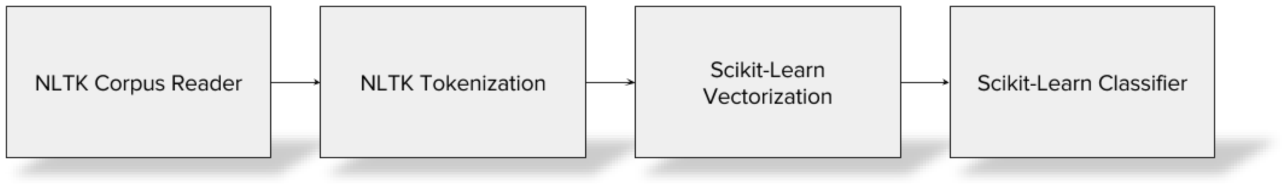
From sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split as tts

IV. PLAN

Pipelining feature of Scikitlearn is a prominent method to build various Machine Learning applications. Scikitlearn's API has two primary interfaces: TRANSFORMER and ESTIMATOR.

Both the transformers and estimators cover a fit() method to adapt internal parameters on the dataset that is used to train the application. Then, the transform() method is performed to modify the data and feature extraction. Finally, the predict() method of estimators is used to generate new data from feature vectors.



- The CorpusReader reads files one at a time off a structured zipped file on the disk and acts as the source of the data.
- The NLTK Tokenizer splits raw text into sentences, words, and punctuation; tags their part of speech and lemmatizes them using the standard lexicon.
- The vectorizer encodes the tokens in the document as a feature vector.
- The classifier is trained by the documents and their labels, pickled to disk and used to make predictions in the future (testing).

V. PREPROCESSING

(Function used in the code is NLTKPreprocessor)

In order to limit the number of features, as well as to provide a highquality representation of the text, I use NLTK's advanced text processing mechanisms including the Punkt_segmenter and tokenizer, the Brill tagger, and lemmatization using the WordNet lexicon.

This not only reduces the vocabulary (and therefore the size of the feature vectors), it also combines redundant features into a single token.

- ✓ First when this transformer is initialized, it loads a variety of corpora and models for use in tokenization. By default, the set of English stop words from NLTK is used, and the WordNetLemmatizer looks up data from the WordNet lexicon.
- ✓ Next, we have the Transformer interface methods: fit, inverse_transform, and transform. The first two are simply pass throughs since there is nothing to fit on this class, nor any ability to do inverse transform. The transform method takes a list of documents (given as the variable, X) and returns a new list of tokenized documents, where each document is transformed into list of ordered tokens.
- ✓ The tokenize method breaks raw strings into sentences, then breaks those sentences into words and punctuation, and applies a part of speech tag. The token is then normalized: made lower case, then stripped of whitespace and other types of punctuation that may be appended. If the token is a stopword or if every character is punctuation, the token is ignored. If it is not ignored, the part of speech is used to lemmatize the token, which is then yielded.
- ✓ Lemmatization is the process of looking up a single word form from the variety of morphologic affixes that can be applied to indicate tense, plurality, gender, etc. First, we need to identify the WordNet tag form based on the Penn Treebank tag, which is returned from NLTK's standard pos_tag function. We simply look to see if the Penn tag starts with 'N', 'V', 'R', or 'J' and can correctly identify if it's a noun, verb, adverb, or adjective.

VI. BUILDING AND EVALUATION

(Function used in the code build_and_evaluate)

The next stage is to create the pipeline, train a classifier, then to evaluate it.

1. The model is split into a training and testing set by shuffling the data
2. The model is trained on the training set and evaluated on testing.
3. A new model is then fit on all of the data and saved to disk (model.pickle).

The function times the build process, evaluates it via the classification report that reports precision, recall, and F1. Then builds a new model on the complete dataset and writes it out to disk.

Output is as follows:

```
Building for evaluation

Evaluation model fit in 112.342 seconds

Classification Report:


precision recall f1 support


neg 0.84 0.84 0.86 193

pos 0.85 0.85 0.85 207

avg / total 0.84 0.84 0.85 400


Building complete model and saving ...

Complete model fit in 119.402 seconds

Model written out to model.pickle
```

VII. PRINTING WORD RATING

(Function used in the code is show_most_informative_features)

In order to use the model, you just built, you would load the pickle from disk and use it's predict method on new text.

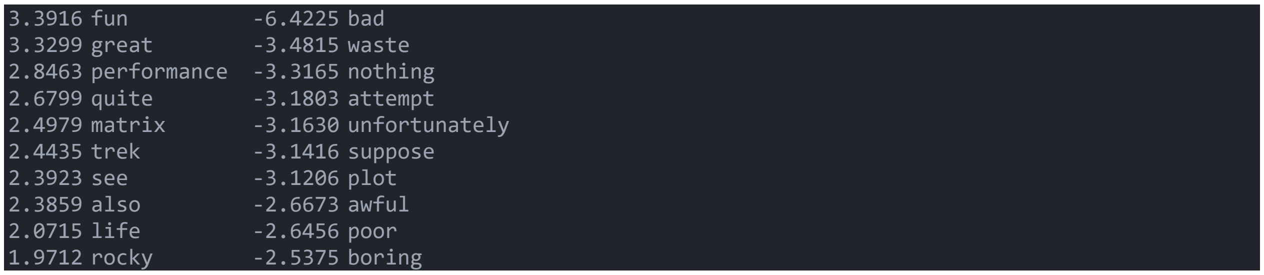
```
yhat = model.predict([
       "This is one of the most amazing movies I have ever seen!",
       "The movie could have been better at times with performance!"
])
```

In order to better understand how our linear model makes these decisions, we can use the coefficients for each feature to determine its weight in terms of positivity and negativity. We can also vectorize a piece of text and see how it's features inform the class decision by multiplying its vector against its weights.

In order to set the number of positive and negative features to print,

Set "n" in the function's parameters.

SAMPLE OUTPUT –



```
3.3916 fun          -6.4225 bad
3.3299 great        -3.4815 waste
2.8463 performance  -3.3165 nothing
2.6799 quite        -3.1803 attempt
2.4979 matrix       -3.1630 unfortunately
2.4435 trek         -3.1416 suppose
2.3923 see          -3.1206 plot
2.3859 also         -2.6673 awful
2.0715 life         -2.6456 poor
1.9712 rocky        -2.5375 boring
```

VIII. CONCLUSION

There are great tools for doing machine learning, topic modeling, and text analysis with Python: Scikit_Learn and NLTK respectively. My approach was to leverage the API model of Scikit_Learn to build Pipelines of transformers that took advantage of other libraries.